SQL and NoSQL Databases

This document provides a brief overview of SQL and NoSQL to help you decide which type of database is right for you. Both types of databases have their benefits and drawbacks, but deciding which to use depends on the type of project you're working on.

SQL

Overview

Structured Query Language (SQL) is a standard language that stores, updates, and retrieves data. SQL is the primary interface for communicating with *"relational databases,"* which store and provide access to data fields that are related to one another. Relational databases are expressed as tables and you can store data in the rows and columns.

SQL transactions adhere to four guidelines, known as **ACID properties,** to help ensure security and stability:

- **Atomicity:** All changes to a transaction are completed at the same time, or not at all.
- **Consistency:** Transactions must always follow the rules set by the database.
- **Isolation:** Concurrent transactions must process separately from one another.
- **Durability:** In the event of unexpected failure, the last known state must be recovered.

Advantages

One of the main advantages of using a SQL database is the accessibility of data to users. No coding skills are required to understand or use the language to access the relational database. SQL utilizes simple syntax keywords such as SELECT, INTO, FROM, UPDATE, DELETE, INSERT, and WHERE to access data from a database. Let's say you want to find the price for a particular product from your database. Your input might look something like:

```
SELECT id, name, price FROM products
```

With this query, you're telling the program to look through the table and retrieve specific data fields from the product table's **ID**, **name**, and **price** columns. SQL databases are generally used in programs found on personal computers, servers, and sometimes even mobile phones. With these portable devices, SQL can retrieve and process large amounts of data with great efficiency. SQL has been widely used for decades, so its standards are clearly defined.

Disadvantages

Horizontal scaling, the ability to use multiple servers to bring the data together into one database, can be challenging for SQL databases. Relational databases depend more on *vertical scaling,* the ability to add more processing power to a single machine. Upgrading to a larger server can get expensive. Additionally, large-scale SQL databases require more time to get set up and need to have full time administrators to keep them running.

NoSQL

Overview

"Not only SQL" (NoSQL), or *non-relational* databases, can store data in formats other than relational tables. Depending on the type of data you want to use, NoSQL offers several database options. Some examples of non-relational databases include graphs, columns, key-values, and documents.

If you look back at the previous example from the SQL Advantages section, a NoSQL database handles the situation differently. Using a non-relational database, you store each data field of the product as an attribute in a single document. The **ID, name,** and **price** are expressed as fields, and the details of the product are associated with each field:

Product_id: "1234567", Product_name: "iPhone", Product_price: "$399",

Advantages

NoSQL databases are easier and cheaper to scale horizontally, allowing them to deliver lower-latency, higher-quality performance. Non-relational databases can store large volumes of unstructured data, and developers can choose from several data formats, depending on the needs of the project. Overall, NoSQL is flexible and capable of handling many changes over time.

Disadvantages

NoSQL databases are less mature than SQL databases, which means it can be difficult to find tools and support if you have issues that need troubleshooting. NoSQL relaxes ACID properties, which provide important protections, in favor of flexibility. This means that non-relational databases might sacrifice security and consistency.

When to use SQL / NoSQL

If your data is structured and the structure rarely changes, or if your application depends heavily on ACID compliance, relational databases might be right for you. Banking systems, for example, must be able to process financial transactions between accounts at the same time, or not at all.

Consider using NoSQL when working with larger projects that contain many, constantly changing variables. If speed is more important to you than ACID compliance and your application needs to run multiple queries at the same time, you might consider using non-relational databases. Common NoSQL use cases include:

- **Graph databases:** Fraud prevention.
- **Column databases:** Web analytics and analyzing data from sensors.
- **Key-value databases:** Game leaderboards and shopping-cart data.
- **Document databases:** Customer data, user-generated content, and order data.